PH-GY 8013 / PH-UY 4603

Software engineering for scientific computing

# Final Project

Due: 18 May 2023, 2:55pm

For the final project, you will design and build a substantial piece of software (primarily in Python — additional support languages OK) to accomplish some task in scientific computing. Each project will involve a group of 3 to 6 students (teams of 4 or more preferred) and can cover any topic of interest to you, possibly related to the research of one or more members of the group.

## Learning objectives

- To gain experience developing a single piece of software *in a group*, along with everything that entails:
  - forethought into design of the code, particularly class hierarchies and interfaces;
  - managing the social and technical aspects of maintaining a common repository;
  - designating different team members as "maintainers" for different portions of the code, and executing a sensible development workflow (probably some form of continuous integration, a.k.a CI, practicing code review, etc);
  - writing tests for portions of code that you directly wrote and that you didn't directly write;
  - setting up automated execution of testing;
  - managing the composition of a collaborative document that is kept under version control;
  - writing a software manual with the help of automatic documentation tools;
  - profiling hotspots in scientific code and (for grad students) speeding it up and/or improving memory usage;
  - etc.
- To try unfamiliar things with which you'd like to gain experience (new languages, algorithms, programming structures, etc);

The end result should have some intrinsic interest and potential utility, even if only to yourself and your research workflow.

## Instructor guidance

Teams should *not* embark on potentially time-consuming paths of work without consulting with me for guidance (use EdSTEM). I will guide you through stages of the project process and provide feedback and advice as needed.

## Scope

The scope of the project should reflect the fact that the project accounts for half the course grade for 3–6 people but also take into account the limited time left in the semester. Code output, or even what the code does, is less important than the items mentioned above (haivng a reasonable design that includes forethought; practicing good git workflows; implementing testing and CI; etc). Your topic should have enough substance and complexity that employing the software engineering elements listed above makes sense, but the functional scope itself can be quite limited. I will help you calibrate to the appropriate level. Project scope should be roughly proportionate to team size.

> chew. I will help you adjust your scope appropriately, and it can shift as the project progresses. Intermediate goals or milestones are helpful; structuring the project as an all-or-nothing "big bang" that doesn't work at all until the last minute when it all (you hope) finally works is probably not a good approach.

## Roll-your-own versus off-the-shelf

You are welcome to use components and code from elsewhere: existing numerical libraries, open-source projects, existing codes from your own research group, etc. However, the overall design and the bulk of the work have to be original.

Building on what others have done is fine (even encouraged, since it avoids re-inventing the wheel and can give you practice in conforming to an existing library's interface or wrapping elements of a library so it can conform to your interface). Just be sure to identify what you have used and where it came from, and get confirmation from the instructor in advance that the scope of work is appropriate.

## Specific requirements

The final project must produce a reliable piece of research software that has been designed with considerable forethought. The following are the core requirements (some pertain to the final product, some to the development process):

**Version control** the code along with all documentation & tests must be kept under version control and hosted in a Github repository (depending on what Git team workflow you decide on, make sure the instructor also read/write access to any extra repos you use). The team is expected to have followed a reasonable team workflow regarding version control. With the possible exception of images and records of profiling results, *everything* must be able to be built from source:

the software, the software manual/documentation, the final report. The test suite should be able to be run automatically, as well (see below).

**Testing + Continuous Integration ((CI)** you should provide a suite of tests (most, if not all, of which can be run automatically) and verify that your software is working the way it is supposed to. Moreover, you must have a continuous integration system (Github Actions is simplest) plugged into that repository that, at a minimum, automates most of the testing.

**Documentation** you must have a software manual that can be output to both PDF format and HTML/web format, most (if not all) of which should be auto-generated from annotations in the source code via a tool like Doxygen or Sphinx (probably Sphinx). Do not hand in the manual – I need to be able to build it from source files in the repository, following clear basic instructions included in your top-level `README`.

**Profiling** once your code is running, you must go through at least one round of profiling — identifying performance bottlenecks in your code and possibly places where the code uses memory inefficiently — following by at least one round of improving the code's performance in some way (memory usage, execution time, etc). Please do not create an artificially bad outcome just so you have an obvious "hotspot" that can be trivially corrected (e.g. by having a Python `for` loop that runs a million times over a NumPy array instead of leveraging NumPy's vectorized operations). The point here isn't to check off a requirement box for the project — you're trying to gain experience with real-world software considerations.

**Final report** in addition to the software manual, each team must submit a report describing the background science, the project goals, the design considerations and ultimate software architecture, the development process (reporting any hiccups and learning experiences encountered along the way), your profiling/tuning results, the outcome of your effort to improve performance in your code (in terms of speed, memory usage, execution time, etc), *lessons learned* (report wrong turns, mini-crises, etc), and any ideas/hopes you might have for future work on the code.

**Final presentation** if we can find a suitable time, I may ask teams to deliver an ∼20 minute presentation to your classmates detailing the above (warts and all). By all means, report on any wrong turns, problems and difficulties that arose with the process, etc. If this happens, it will probably be before the final project is due and will likely describe "work in progress".

## Deliverables

To summarize/reiterate what is above, the deliverables for the project will be:

- a piece of research software (hosted in a Github repository with a CI framework plugged in to help manage the testing);

- a software manual in both PDF format and HTML/web format, that can be built from source (instructions for how to do this should be in the README of the Github repo);
- a PDF report (that can also be built from source files in the repo) describing:
  - the background science;
  - the project goals;
  - the design considerations and final software architecture (including any pertinent UML diagrams, i.e. an update of the design document);
  - the development process (including recounting any hiccups encountered and lessons learned along the way);
  - your profiling results, and the outcome of your effort to improve performance in your code (speed, memory usage, execution time, etc);
- (MAYBE) a presentation to your classmates detailing the above (warts and all).

Remember, you are being assessed on the *process* you follow as more than on "impressiveness" of the final product. It is better to have a project with a very narrow scope where the development milestones listed above are followed than some more ambitious project in which the version control workflow breaks down and people are emailing source code files back and forth because a deadline is looming. The latter will lead to a worse final grade. If you hit any sort of roadblocks, in terms of process or design or workflow or whatever, seek help early (from me, from classmates — use EdSTEM). Don't spin your wheels for days or weeks.

### A note about document preparation

Technically, the only real deliverable is the Github repo for the project – the software manual and final report must also be kept as versioned source files in the repo, and the repo should have a means to build the manual and report from source (e.g. a Makefile — we will discuss this in class). In particular...

> **You *CANNOT* use MS Word or Google Docs or other word processors for the final project.** You'll need to use some sort of plain-text markup language[1] (LaTeX or Markdown or reST, etc).

[1] If you've never heard of `pandoc` (see its github repo), you might want to look into it. Not necessarily for your project (though you're welcome to use it for that if you like), but more just as a Swiss-army knife for markup languages that can facilitate storing your research life in plain-text. I'm not saying you *should* use `pandoc` — reST or direct LaTeX might be better — I'm merely making you aware of another tool.

### A note about planning

> mess things up as you go, particularly if this is your first experience working on a software project with people you may not know well and if you're relatively new to Git. Factor in time for that, and err on the side of communicating more rather than less with me. I'll be here to help.

## Design document

One of the first requirements (which will be a homework assignment) will be a design document, a fairly detailed description of the project that includes:

- background on the topic
- an overview of what your code will do
- the basic organization and components, with accompanying UML diagram(s)
- a discussion of interfaces, both between different parts of the code and between the code and users
- a rough timetable of what you hope to get done by when and who will be doing what (these are just initial forecasts and task assignments – they will almost certainly change once you get going).

The design document should describe any "roll-your-own-versus-off-the-shelf" decisions, and it should describe (and ideally provide links to) any code you intend to use from elsewhere, whether libraries or earlier incarnations of code that may belong to members of your group.

The design document doesn't need to be more than a few pages (a few examples from prior times I have done a similar course can be found on EdSTEM). And it's more of a draft rather than something set in stone.

The intended audience is both the instructor *and* your classmates – we will have a feedback session during class when we all comment on one another's design documents, sharing ideas and suggestions.

## Schedule and deliverables

The table below lists the major dates and milestones for the project, followed by a more detailed description of each.

| Date | Item |
|---|---|
| 21 Apr | Choose partners, team name, & have a rough proposal |
| 25 Apr | Design document due |
| 27/8 Apr | Design reviews (over Zoom) |
| 30 Apr | Git + CI setup |
| 10 May | Alpha version |
| 18 May | Projects due |
| ?? May | Project presentations |

**21 Apr: Finalize teams and the general topic/proposal.** I'll make an Ed-STEM thread to which every team can reply with: a team name, list of team members (names and NetIDs), and a rough proposal. Talk to me if you need help crystallizing this or figuring out what's a reasonable scope vs. too much or too little (or figuring out a project at all). Both before and after this date, I will provide feedback on the scope of your proposal and iterate with you to help flesh it out.

**25 Apr: Design document due** (as discussed above). All the design documents will be shared with all the other groups, to prepare for the design review. No MS Word, no Google Docs — please use a plain-text markup language for this.

**27/8 Apr: Design reviews.**

This is an opportunity to crystallize the idea and get feedback from me and one another on scope and overall design. Everyone will likely have overlapping and related questions and challenges in their own designs, so plase attend this session once we schedule it.

**30 Apr: Git setup.** By this date, your team should have come to a decision what the team Git workflow will be (the discussion of Git workflows in lecture will inform this, but you can adopt something other than the workflows we end up discussing in class) and should have designated one "main" repository for the project (if that repo is private, make sure the instructor has access). If you're going to use anything other than Github Actions for CI, you should also have the CI system setup and integrated with the repo by this date.

**10 May: Alpha version.** By this date (just after Thanksgiving break), you should have (and be able to demonstrate) an almost-working version of the core functionality of your project (i.e. basic operations that form the essence of your project). Many features may still need to be implemented, but both you and I should be convinced that the project can be completed. Communicate with me if you think you need to reduce scope.

**18 May: Projects due.** Everything should be handed in by 11:55pm (I'll evaluate the latest repo commit prior to this time). In addition to the code itself, I should be able to build the software manual and final report from source.

**?? May: Project presentations.** Exact date TB